ELEC4605 Assignment

Corey Anderson

Friday 1st November, 2024

x	Name	Phone Number
0	H	4
1	G	3
2	E	1
3	F	2
4	C	6
5	A	0
6	D	5
7	B	7

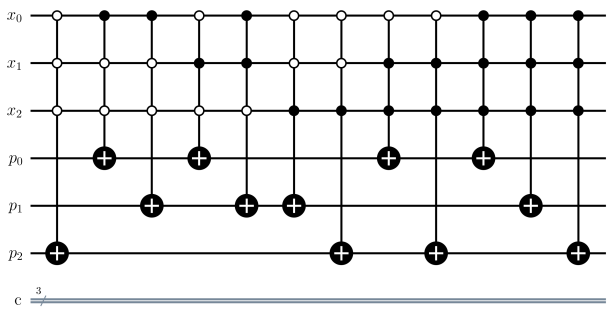
Table 1: My Randomized Quantum Memory

a) **What is the minimum number of qubits (m) that you will require in each of the four registers to implement this algorithm?**

$$m = \lceil \log_2(8) \rceil = 3 \tag{1}$$

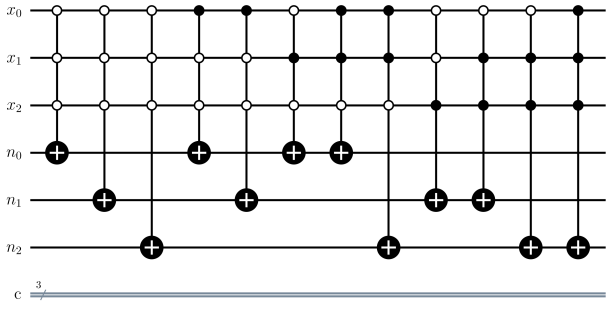
b) **Draw a circuit that implements the quantum memory unitary U_{QMP}, i.e. one which writes into the register $|p\rangle$ the random phone number stored in the element identified by $|x\rangle$.**

To convert the qubits $|x\rangle$ to $|p\rangle$, I used a series of Toffoli gates with 3 controls. Each gate acts to flip a qubit for a given input. This covers all input possibilities and maps them to the desired outputs.



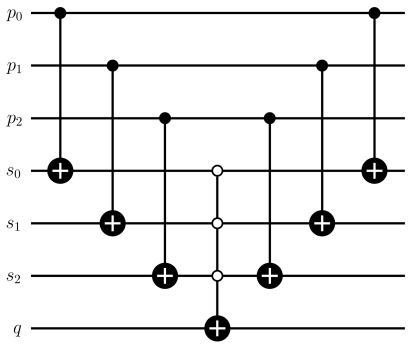
c) **Draw a circuit that implements the quantum memory unitary U_{QMN}, i.e. one which writes into the register $|n\rangle$ the random name stored in the element identified by $|x\rangle$.**

The implementation of this is very similar to the previous part, but for a different set of qubit mappings.



d) **Draw a circuit that implements the oracle unitary, i.e. one which marks the correct solution with an inverted phase. The correct solution is the one where $|s\rangle = |p\rangle$. Be careful not to alter the state of $|p\rangle$ in this step, as the step must be repeated several times as part of the Grover iteration.**

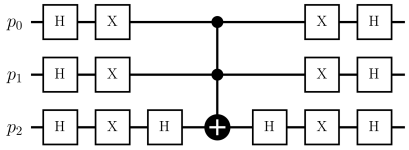
The output of each CNOT gate is $|p_i \oplus s_i\rangle$, which is 0 if $|p_i\rangle = |s_i\rangle$. This is why I have used a 3 qubit controlled Toffoli gate, detecting for the 0 (equal) condition in order to mark the unitary with the inverted phase. The circuit is symmetric as to not alter the $|p_i\rangle$ qubits, which allow me to run multiple iterations of the Grover algorithm.



e) **Draw a circuit that implements the inversion about the mean unitary. Make sure that this circuit is efficient (i.e. that it contains a number of steps that is polynomial in the number of qubits).** Ignoring the outer Hadamards, the circuit inverts the phase of the $|000\rangle$ state:

$$|000\rangle \to |111\rangle \to |11\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \to |11\rangle \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) \to -|111\rangle \to -|000\rangle \tag{2}$$

It is similarly shown that all other states are not affected. The operation of this is $2|0\rangle\langle 0| - I$. Combined with the outer Hadamards give $H^{\otimes 3}(2|0\rangle\langle 0| - I)H^{\otimes 3} = 2|\psi\rangle\langle\psi| - I$, which is the required inversion about the mean operation.

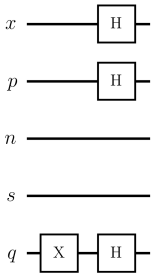


f) **Draw the circuit needed to initialize the qubit registers qubit $|x\rangle$, $|n\rangle$, $|p\rangle$ and $|s\rangle$ and the oracle qubit $|q\rangle$.**

We place the Hadamard in front of the $|x\rangle$ qubit lines in order to consider all possible combinations. The reason we need the Hadamard in front of the $|p\rangle$ lines is to avoid entanglement. Suppose there were no Hadamard in front of the $|p\rangle$ lines. The Toffoli gates performs a function $f(x)$, which maps $|x\rangle$ to $|f(x)\rangle$ in the p register :

$$U_{QMP}|x\rangle|p\rangle = |x\rangle|p \oplus f(x)\rangle = |x\rangle_x |f(x)\rangle_p \text{ ,since } p \text{ is in the } |0\rangle \text{ state.} \tag{3}$$

However, this means that if we know $|p\rangle = |f(010)\rangle_p$, then the superposition of the x register collapses to a single state $|x\rangle = |010\rangle$. This loss of superposition means that our circuits that depend on x after will not function correctly. Thus, we put Hadamards on the p qubit lines to initialize. Finally, we want the oracle qubit to be initialized to $|q\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$ using X and H gates, in order to perform the phase inversion of the correct solution.

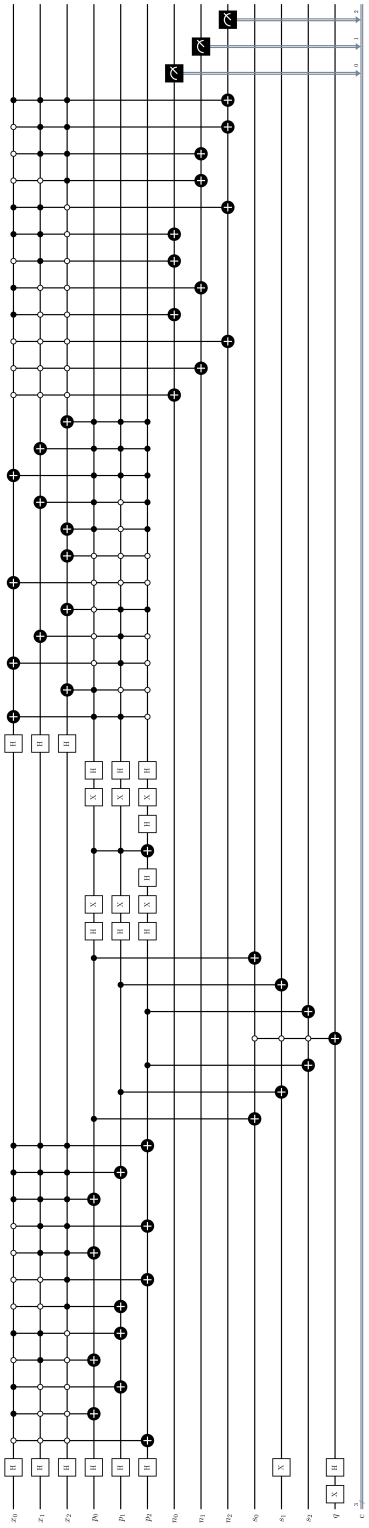


g) **Put parts b-f together and draw the entire circuit that implements the quantum search algorithm. At the output of the circuit, you should obtain the name of the person belonging to the phone number s.**

The circuit functions as follows:

Map x to p registers using $U_{QMP} \implies$ Perform Grover iterations to focus the correct $|p_s\rangle \implies$ Convert that $|p_s\rangle$ back into $|x_s\rangle \implies$ Map $|x_s\rangle$ into $|n_s\rangle$ using $U_{QMN} \implies$ Measure!

Note that there are additional H gates before the conversion from $|p_s\rangle \rightarrow |x_s\rangle$ because we are now dealing with a single state which is the correct state, so we no longer want a superposition in the x registers. This circuit is only for 1 iteration of the Grover's Algorithm. For more iterations, just duplicate the circuits found in parts d) and e).



h) **Implement the entire algorithm in code and demonstrate that it works by running the search for a few (at least 3) random phone numbers (e.g. $|s\rangle = |2\rangle, |7\rangle, |5\rangle$).**

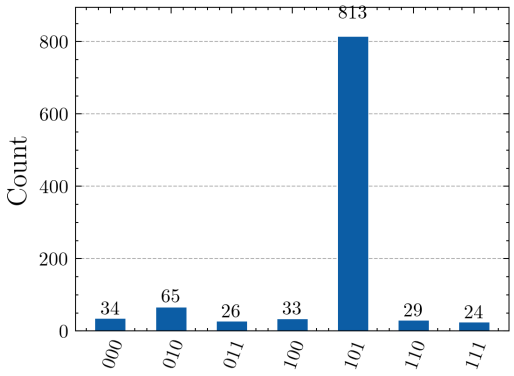
Below is the comparison between the outputs of the 1 iteration and 2 iteration Grover algorithm. According to this, the accuracy of the algorithm as a function of iteration number oscillates. The iteration number for the peak accuracy is found using

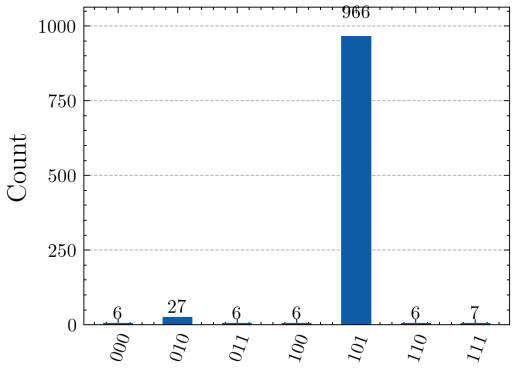
$$k_{\text{optimal}} = \left\lfloor \frac{\pi}{4}\sqrt{8 \text{ entries}} \right\rfloor = 2 \text{ repetitions} \tag{4}$$

1 Iteration

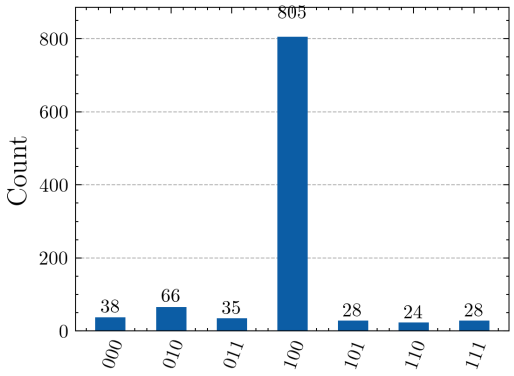
2 Iteration

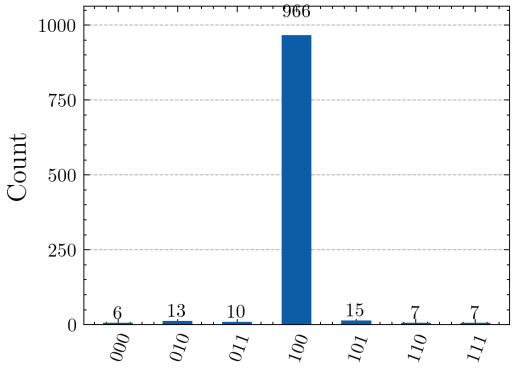
- $|s\rangle = |2\rangle = |010_2\rangle$



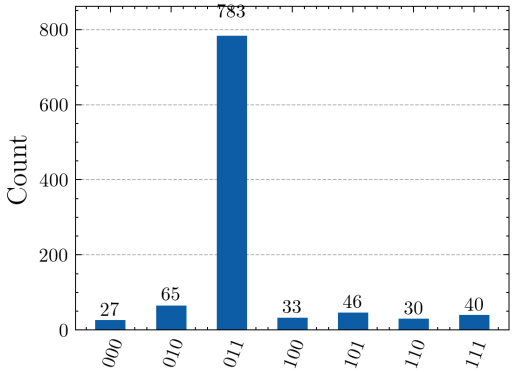


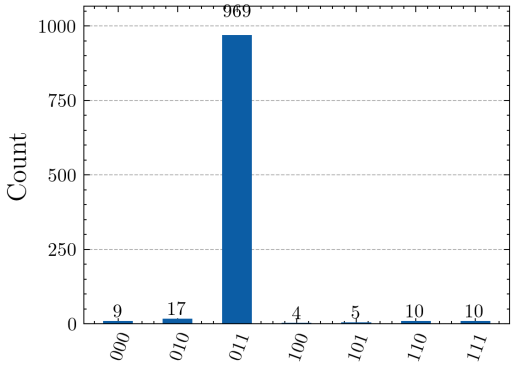
- $|s\rangle = |7\rangle = |111_2\rangle$





- $|s\rangle = |5\rangle = |110_2\rangle$





A Code Appendix

A.1 U_{QMP} Circuit

```
x = QuantumRegister(3, 'x')
p = QuantumRegister(3, 'p')
c = ClassicalRegister(3, 'c')
qcp = QuantumCircuit(x,p,c)

# Case 0: 000 -> 100
qcp.append(MCXGate(3, ctrl_state="000"), [0, 1, 2, 5])

# Case 1: 001 -> 011
qcp.append(MCXGate(3, ctrl_state="001"), [0, 1, 2, 3])
qcp.append(MCXGate(3, ctrl_state="001"), [0, 1, 2, 4])

# Case 2: 010 -> 001
qcp.append(MCXGate(3, ctrl_state="010"), [0, 1, 2, 3])

# Case 3: 011 -> 010
qcp.append(MCXGate(3, ctrl_state="011"), [0, 1, 2, 4])

# Case 4: 100 -> 110
qcp.append(MCXGate(3, ctrl_state="100"), [0, 1, 2, 4])
qcp.append(MCXGate(3, ctrl_state="100"), [0, 1, 2, 5])

# Case 5: 101 -> 000

# Case 6: 110 -> 101
qcp.append(MCXGate(3, ctrl_state="110"), [0, 1, 2, 3])
qcp.append(MCXGate(3, ctrl_state="110"), [0, 1, 2, 5])

# Case 7: 111 -> 111
qcp.append(MCXGate(3, ctrl_state="111"), [0, 1, 2, 3])
qcp.append(MCXGate(3, ctrl_state="111"), [0, 1, 2, 4])
qcp.append(MCXGate(3, ctrl_state="111"), [0, 1, 2, 5])

qcp.barrier()
# qcp.measure([3, 4, 5], [0, 1, 2])
# Create the drawing without displaying it
qcp.draw(output='mpl', plot_barriers=False, style='bw')
```

A.2 U_{QMN} Circuit

```
x = QuantumRegister(3, 'x')
n = QuantumRegister(3, 'n')
c = ClassicalRegister(3, 'c')
qcn = QuantumCircuit(x,n,c)

# Case 0: 000 -> 111
qcn.append(MCXGate(3, ctrl_state="000"), [0, 1, 2, 3])
qcn.append(MCXGate(3, ctrl_state="000"), [0, 1, 2, 4])
qcn.append(MCXGate(3, ctrl_state="000"), [0, 1, 2, 5])
# Case 1: 001 -> 110
qcn.append(MCXGate(3, ctrl_state="001"), [0, 1, 2, 3])
qcn.append(MCXGate(3, ctrl_state="001"), [0, 1, 2, 4])

# Case 2: 010 -> 100
qcn.append(MCXGate(3, ctrl_state="010"), [0, 1, 2, 3])

# Case 3: 011 -> 101
qcn.append(MCXGate(3, ctrl_state="011"), [0, 1, 2, 3])
qcn.append(MCXGate(3, ctrl_state="011"), [0, 1, 2, 5])
# Case 4: 100 -> 010
qcn.append(MCXGate(3, ctrl_state="100"), [0, 1, 2, 4])

# Case 5: 101 -> 000

# Case 6: 110 -> 011
qcn.append(MCXGate(3, ctrl_state="110"), [0, 1, 2, 4])
qcn.append(MCXGate(3, ctrl_state="110"), [0, 1, 2, 5])

# Case 7: 111 -> 001
qcn.append(MCXGate(3, ctrl_state="111"), [0, 1, 2, 5])

qcn.barrier()
# qcp.measure([3, 4, 5], [0, 1, 2])

qcn.draw(output='mpl', plot_barriers=False, style='bw')
```

A.3 Oracle Circuit

```
p = QuantumRegister(3, 'p')
s = QuantumRegister(3, 's')
q = QuantumRegister(1, 'q')
qcs = QuantumCircuit(p,s,q)

qcs.append(MCXGate(1, ctrl_state="1"), [0, 3])
qcs.append(MCXGate(1, ctrl_state="1"), [1, 4])
qcs.append(MCXGate(1, ctrl_state="1"), [2, 5])


qcs.append(MCXGate(3, ctrl_state="000"), [3, 4, 5, 6])

qcs.append(MCXGate(1, ctrl_state="1"), [2, 5])
qcs.append(MCXGate(1, ctrl_state="1"), [1, 4])
qcs.append(MCXGate(1, ctrl_state="1"), [0, 3])


fig = qcs.draw(output='mpl', style='bw')
fig.savefig("qcs.png", dpi=300)
```

A.4 Mean Inversion Circuit

```
p = QuantumRegister(3, 'p')
qci = QuantumCircuit(p)

qci.h([0,1,2])
qci.x([0,1,2])
qci.h(2)
qci.append(MCXGate(2, ctrl_state="11"), [0, 1, 2])
qci.h(2)
qci.barrier()
qci.x([0,1,2])
qci.h([0,1,2])

fig = qci.draw(output='mpl', plot_barriers=False, style='bw')
fig.savefig("qci.png", dpi=300)
```

A.5 Initialization Circuit

```
x = QuantumRegister(3, 'x')
p = QuantumRegister(3, 'p')
n = QuantumRegister(3, 'n')
s = QuantumRegister(3, 's')
q = QuantumRegister(1, 'q')
qca = QuantumCircuit(x, p, n, s, q)

qca.x(12)
qca.barrier()
qca.h(range(6)) # 6
qca.h(12)


qca.draw(output='mpl', plot_barriers=False, style='bw')
```

A.6 Reverse U_{QMP} Circuit

```
x = QuantumRegister(3, 'x')
p = QuantumRegister(3, 'p')
c = ClassicalRegister(3, 'c')
qcp_inv = QuantumCircuit(x,p,c)

# Case 0: 100 -> 000

# Case 1: 011 -> 001
qcp_inv.append(MCXGate(3, ctrl_state="011"), [3, 4, 5, 0])

# Case 2: 001 -> 010
qcp_inv.append(MCXGate(3, ctrl_state="001"), [3, 4, 5, 2])

# Case 3: 010 -> 011
qcp_inv.append(MCXGate(3, ctrl_state="010"), [3, 4, 5, 0])
qcp_inv.append(MCXGate(3, ctrl_state="010"), [3, 4, 5, 1])

# Case 4: 110 -> 100
qcp_inv.append(MCXGate(3, ctrl_state="110"), [3, 4, 5, 2])

# Case 5: 000 -> 101
qcp_inv.append(MCXGate(3, ctrl_state="000"), [3, 4, 5, 0])
qcp_inv.append(MCXGate(3, ctrl_state="000"), [3, 4, 5, 2])

# Case 6: 101 -> 110
qcp_inv.append(MCXGate(3, ctrl_state="101"), [3, 4, 5, 2])
qcp_inv.append(MCXGate(3, ctrl_state="101"), [3, 4, 5, 1])

# Case 7: 111 -> 111
qcp_inv.append(MCXGate(3, ctrl_state="111"), [3, 4, 5, 0])
qcp_inv.append(MCXGate(3, ctrl_state="111"), [3, 4, 5, 1])
qcp_inv.append(MCXGate(3, ctrl_state="111"), [3, 4, 5, 2])

qcp_inv.barrier()

fig = qcp_inv.draw(output='mpl', plot_barriers=False, style='bw')
fig.savefig("qcp_inv.png", dpi=300)
```

A.7 Entire Circuit and Simulation

```
# U_qmp
qc = qca.compose(qcp, [0, 1, 2, 3, 4, 5])
qc.x(10) # s012 = 9, 10, 11

qc.barrier()

# Oracle
qc = qc.compose(qcs, [3,4,5,9,10,11,12])
qc.barrier()

# Invert about Mean
qc = qc.compose(qci, [3,4,5])
qc.barrier()

# Oracle
qc = qc.compose(qcs, [3,4,5,9,10,11,12])
qc.barrier()

# Invert about Mean
qc = qc.compose(qci, [3,4,5])
qc.barrier()

# Remove superposition (reinitialise to x=000)
qc.h([0,1,2])

qc = qc.compose(qcp_inv, [0,1,2,3,4,5])

qc = qc.compose(qcn, [0,1,2,6,7,8])

qc.measure(n, c)

qc.draw(output='mpl', plot_barriers=False, style='bw', fold=56)


# SIMULATION
simulator = AerSimulator()
qct = transpile(qc, simulator)

result = simulator.run(qct).result()
counts = result.get_counts(qct)

plot_histogram(counts, figsize=(4, 3))
```